Open in app ↗ Sign up Sign in

Medium Search

Defi + Uniswap + Ethereum + Finance + Volatility +

Designing a constant volatility AMM

9 min read · Apr 4, 2022

Guillaume Lambert Follow

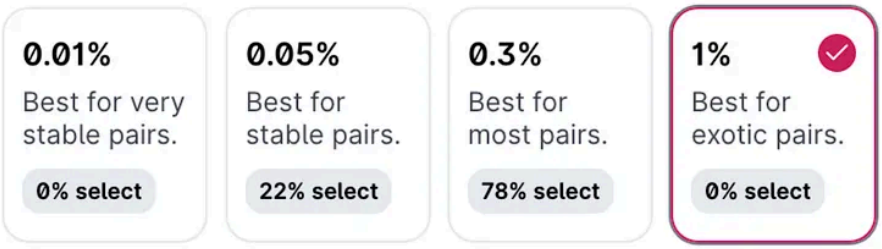
Listen Share

- *TL;DR: Constant volatility AMMs requires the fee rate to change at every txn.*
- *By computing the fee on a per-trade basis in Uniswap, it is possible to target a constant volatility (eg. 100% annualized) regardless of the actual price dynamics of the underlying assets.*
- *Trading assets in a constant volatility AMM should feel like a delicate balancing act between slippage & fees.*

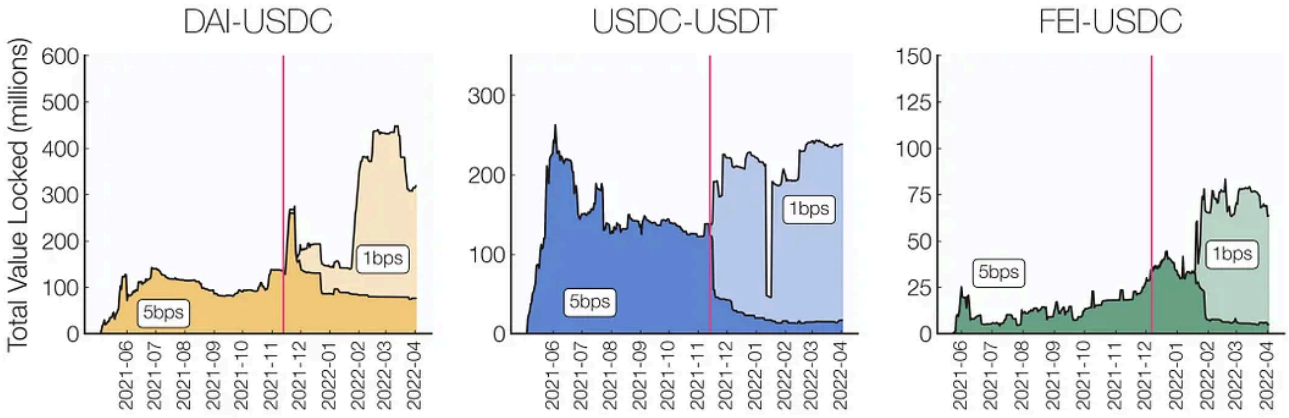
How feeTier impacts trade volumes

In Uniswap v3, users deploying a new pool can specify a feeTier for each asset pair. The feeTier currently available are 1%, 0.3%, 0.05%, and 0.01%.

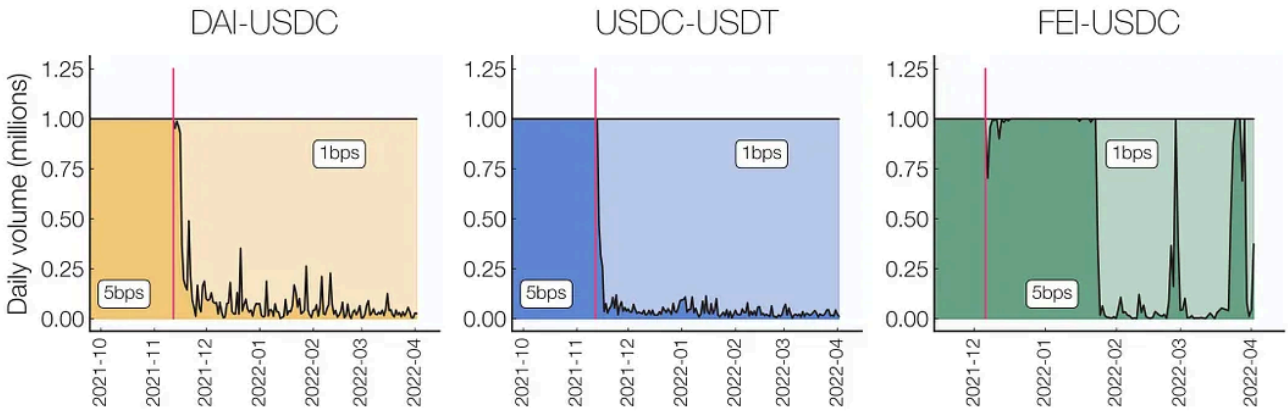
The conventional wisdom is that a higher fee is best for exotic pairs and the 0.01% is more suitable for the most stable of pairs.



How accurate is this “conventional wisdom”. In fact, shortly after the Uniswap governance activated the 1bps fee tier, we immediately saw a redirection of almost all stablecoin-stablecoin TVL and volume from the 0.05% pools to the 0.01% pool. And even though the 0.05% pools still hold about 1/3 of the total TVL for the DAI-USDC pair, 95% of the total volume is routed through the 1bps pool:



Total value locked of the 5bps and 1bps pools. Pink line = 1bps pool creation.



Volume partition between 1bps and 5bps after pool activation

Interestingly, some liquidity was deployed in the ETH-USDC 1bps pool:since the creation of the 1bps fee tier, 34 LP positions have been deployed to the ETH-USDC-1bps pool (total value deployed= $15M):

ETH–USDC 0.01% pool

tokenId	owner	price Lower	price Upper	Amount Deposited	Amount Collected	Profit
153900	0x0ce37f7166d9f94180274348207829f72ce23737	4525.04	4650.73	1212057.00	1218695.94	6638.94
153918	0x0ce37f7166d9f94180274348207829f72ce23737	4551.81	4589.75	671710.00	671838.12	128.12
154960	0x0ce37f7166d9f94180274348207829f72ce23737	4000.55	4250.89	1500.00	1499.15	-0.85
154969	0x0ce37f7166d9f94180274348207829f72ce23737	3800.12	4500.22	2768875.84	2764695.61	-4180.23
154974	0x0ce37f7166d9f94180274348207829f72ce23737	4209.44	6000.31	547259.43	544073.11	-3186.32
155020	0x72a916702bd97923e55d78ea5a3f413dec7f7f85	4174.23	4175.9	5427349.69	5418419.10	-8930.59
159475	0x0be0ecc301a1c0175f07a66243cff628c24db852	4175.48	1000001.32	2072.58	2088.55	15.97
159481	0xd978f6ae3377c1a2bd74741ada0398b7fe16ba01	1.0	1000001.32	1237.54	1219.50	-18.03
161236	0x9e309c439c118804d9c2cf5eac4dfc62cc6f95c2	4303.07	5103.46	12366.71	12275.94	-90.77
164332	0xff1a98f4923cd203bdc5e0631d122fae3e16f541	4291.89	4310.39	330.20	307.50	-22.70
164719	0x97d05a137c2cfa0b9f45976eedd630f95cb691fb	4339.36	4348.05	2283.55	2220.21	-63.34
168474	0xf72b447a0a6f71c70b3345c8ef41cc2980dcf62d	2000.04	6000.31	15298.47	12208.08	-3090.39
170475	0xb99426903d812a09b8de7df6708c70f97d3dd0ae	4001.35	4017.39	5817.23	5828.92	11.68
171848	0xb99426903d812a09b8de7df6708c70f97d3dd0ae	4069.14	4077.28	15331.05	15294.93	-36.12
179672	0x36a48decbafd773c5d1d7b6ecb6615d59dbf14e4	3205.1	3220.2	252801.13	252823.25	22.13
181575	0x8410373df6e9b20765c9599c26d585b2cd0ef628	3200.29	3300.06	794210.89	796997.97	2787.09
181772	0x62cdced524ba6c9d023c0b8682e835af6aba44fc	3132.85	3733.83	64437.31	63435.29	-1002.02
187704	0x84a78c1beada8d1a829531f3299f7a98324f1251	2000.04	10000.01	62395.73	63578.14	1182.40
188241	0x4973cb0b263bb34a1648e821f0df548bcc294614	2402.85	2807.09	1191.98	1234.19	42.21
191061	0xf72b447a0a6f71c70b3345c8ef41cc2980dcf62d	1000.0	7000.66	14859.10	0.00	still open
197200	0x254242457f3f61d9050e451a61a107ea92d5e2b1	750.07	9977.04	50019.07	50638.46	619.40
197310	0xd120cf3e0408dd794f856e8ca2a23e3396a9b687	2722.23	2722.5	886484.39	886472.43	-11.95
197325	0x72a916702bd97923e55d78ea5a3f413dec7f7f85	2705.4	2705.67	2500000.00	2500000.00	-0.00
197580	0x0359f194b3b801a1cce464eb8367a5a7b796a69d	2746.29	2757.29	4.19	0.00	still open
197611	0x81efd4526dfdbbddfe306d04073fc6beff25a1e1	2772.22	2775.83	5.52	0.00	still open
204531	0x3180c4f8e7574a538d2b3121223dfaab05eccc48	FULL	RANGE	60.00	0.00	still open
207738	0x8ec5c5528db3f1c6d8e3c4256e87de04ca7fffdb	2900.69	2963.43	26285.36	25942.79	-342.56
208910	0x4b5e16fc413a2c1c08499df18f6d10e422ed78f5	2395.65	4995.42	23601.71	24413.17	811.47
210886	0x39124fd9c66d71c0e9394df1f63f86e1bc81057c	2985.14	3500.26	53191.54	53582.13	390.58
212153	0xa07161106423845e2a7ff82ab17fc19a1fd1babe	3076.97	3096.1	0.20	0.00	still open
213004	0xc887b3406424838ae6545686027 8a5e6e40fcb08	3154.86	3174.79	0.00	0.00	still open
213137	0x1e30dc44f1a0738bcd74d649ce9e8b56a4bdfd93	FULL	RANGE	1.00	0.00	still open
213187	0x2de2c93d6d78711de0b30fdcc723bf6b4ebe5b58	3164.65	3174.79	4.51	4.50	-0.01
215031	0x5f40324ed97badd9079e67e162c83b1fb9168a12	FULL	RANGE	2.00	0.00	still open

What was the average return of the ETH-USDC-0.01% liquidity providers? Well, not great: the median amount of collected fees is about 0.005% of the deposited amounts. And although some positions ended up with more value in USD terms, most positions gains/losses shown in the table above were due to changes in the price of ETH when the position was burned.

This is an interesting observation: whereas the 0.05% pool is perfectly suitable for several ETH-stablecoin pairs, it seems that the yields for the 0.01% pools are insultingly low.

Why didn't the 1bps fee tier gain traction for non-stablecoin-stablecoin pairs? Is there something special about the 1bps fee tier that makes it non-viable for ETH-based pairs?

Going even deeper, could an AMM with infinite liquidity implement a 0% trading fee? Well it turns out that, for one, a fee equal to zero is especially detrimental because of the path-dependent nature of geometric means (vs. arithmetic means). But Dave White et al showed that anything above zero is close to being optimal:

> *The fee should therefore be set as low as possible without being zero so that rebalances are triggered by increasingly tiny price movements. Dan Robinson calls this "picking up pennies in the quantum foam."*
>
> *However, when the fee is exactly zero, all the benefits of rebalancing disappear, and in most cases LPs are worse off than they would be if they just held the unrebalanced portfolio. source: Uniswap's Financial Alchemy*

How much of that is true? First, let's explore how the feeTier influences the transaction volume and liquidity distribution.

How feeTier, dailyVolume and tickLiquidity impacts volatility

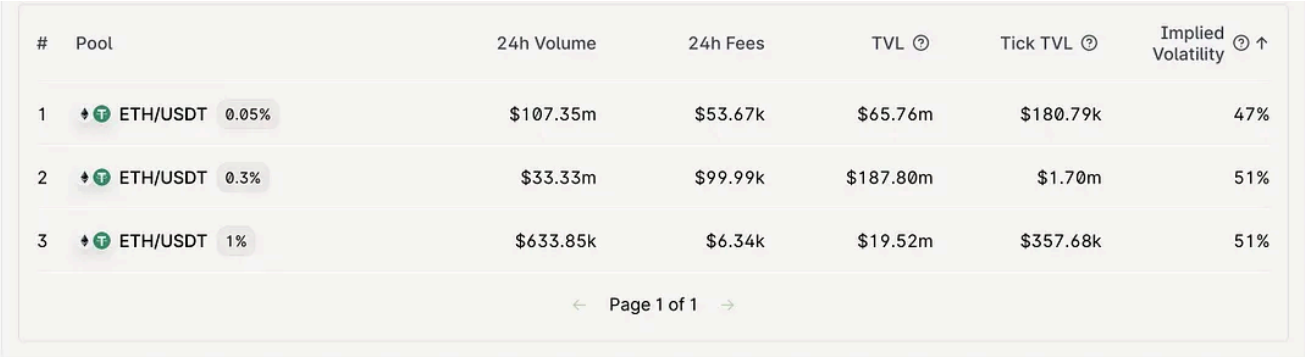
We've derived in a previous post an expression linking the feeTier, the dailyVolume, and the tickLiquidity:

$$\sigma = 2 \cdot \texttt{feeTier} \cdot \sqrt{\frac{\texttt{dailyVolume}}{\texttt{tickTVL}}}$$

In a series of tweets from last weekend, I described how this expression appears to hold across many pools in the Uniswap v3 ecosystem.

Specifically, these posts show that even though the feeTier, the daily volume, and the amount of liquidity locked in a pool do vary quite a bit between pools, it appears that the "market" has figured out a way to distribute that liquidity to maintain a constant volatility between pools.

For instance, the 1%, 0.3%, and 0.05% ETH-USDT pools all have IV equal to approximately 50% even though the volume and TVL vary greatly between pools:



#	Pool	24h Volume	24h Fees	TVL	Tick TVL	Implied Volatility
1	ETH/USDT 0.05%	$107.35m	$53.67k	$65.76m	$180.79k	47%
2	ETH/USDT 0.3%	$33.33m	$99.99k	$187.80m	$1.70m	51%
3	ETH/USDT 1%	$633.85k	$6.34k	$19.52m	$357.68k	51%

Page 1 of 1

I made the argument that the volatility of an asset is an invariant that must remain the same across pools. This is true in part because the Uni v3 Router contract "smartly" only direct an order to the 0.05% pool only if the slippage is less than 0.3%, to the 0.3% pool only if the slippage is less than 1%, and to the 1% pool if the slippage is greater than 1%.

As a result, the 1% pool sees a much lower *number of transactions*, but the slippage will be higher for each one. Similarly, the 0.05% pool is traded much more frequently but the average slippage is lower.

Ultimately, the yield will be the same across fee tiers because average returns are directly related to the volatility of an asset. “Rational” market participants will recognize this will relocate liquidity from low volatility pool (ie. pools with low volume OR too much liquidity) to high volatility pools to maximize their yields.

In the case of the 1bps ETH-USDC pool, the TVL was too low to support any kind of volume: if the 1bps and 5bps were to have the same volume, the TVL of the 1bps pool needs to be 25 times larger than the TVL of the 5bps pool.

Responsive fee: building a constant volatility AMM

What if we let the feeTier float and instead *target a specific volatility* σ?

Get Guillaume Lambert’s stories in your inbox

Join Medium for free to get updates from this writer.

Enter your email

Subscribe

Remember me for faster sign in

In the previous section, we derived an expression for the implied volatility that depends on i) the feeTier, ii) the daily volume, and iii) the at-tick volatility. We made the argument that returns for LPs should be the same across all fee tiers as long as each pool follows the same asset.

We postulate here that we can achieve this by inverting the IV expression so that the feeTier depends on liquidity+volume while keeping the volatility constant:

$$\texttt{feeTier} = \frac{\sigma}{2} \cdot \sqrt{\frac{\texttt{tickTVL}}{\texttt{dailyVolume}}}$$

Basing the fee of the *daily* volume is perhaps a bit arbitrary. What about computing the feeTier on a per-trade basis?

Let's consider a user wants to swap amount0 of token0 for amount1 of token1. First, the dailyVolume would need to be transformed into tradeSize/time. If we set the "time" variable to the time since the last transaction, and the tradeSize as amount0, then the feeTier would be:

$$\texttt{feeTier} = \frac{\sigma}{2} \cdot \sqrt{\frac{\texttt{tickTVL}}{\texttt{amount0}}} \sqrt{\texttt{time since last transaction}}$$

If we want to target a specific volatility, say vol=100% annualized, then we need to convert the value of σ to the same using as "time since last transaction", which is given in seconds, to get:

$$\texttt{feeTier} = \frac{\texttt{IV(per year)}}{2\sqrt{365 \cdot 24 \cdot 60 \cdot 60}} \cdot \sqrt{\frac{\texttt{tickTVL}}{\texttt{amount0}}} \sqrt{\Delta\texttt{T(seconds)}}$$

This expression has very interesting consequences.

1. The more time between transaction, the higher the fee
2. The larger the trade size, the smaller the fee
3. The higher the liquidity at the traded tick, the higher the fee

All points above seem to reduce fees in response to: large trades that happen very frequently, in regions with low liquidity. That'll drive the volatility up for you.

Let's consider an example, where this reactive fee is implemented in the ETH-DAI-0.3% pool. The current value locked at the current tick is 316 ETH and the implied volatility at that tick is 56%.

If one trade happens every block (blocktime=15s) and the pool targets an annualized volatility of 100%, then the purchase of 1 ETH would result in a fee equal to: √15/11231 * √(315/1) = 0.6%. About twice the "normal" fee.

Waiting another block would increase this fee to 0.86%, with a general formula for the 1 ETH transaction being 0.6% * √(no of blocks since last txn).

However, let's say the user is a whale that instead sells 1000 ETH. Then the fee would be 0.02% (!), generating only 0.2ETH in trading fees for the liquidity providers. The slippage on that trade would also be 3 ticks for a ~0.9% slippage, so LPs would only collect 0.067ETH at each tick collectively.

This slippage may be arbitraged back if the new spot price is not aligned with other exchanges'. So in this case, another arbitrage trade may occur to bring the price back to where it was.

Because of this, non-arbitrage trades would occur more frequently because fees are lower for larger trades, and price will need to be arbed back with a correspondingly large trade to reduce gas fees. So even though the effective fee was only 0.02%, a single transaction will create a ripple effect of arbitrage trades that may effectively create more volume than if the pool had a constant fee.

All of these extra trades will contribute to increasing the volatility of the asset to the target rate, regardless of the actual realized volatility of the underlying asset on centralized exchanges.

A constant volatility AMM in Uniswap v2

What if we do not want to use Uniswap v3 with its tick-base liquidity paradigm (maybe in 1 more year we can) and instead use a vanilla x*y/k AMM like Uniswap v2?

In that case, the tickTVL is meaningless because liquidity in Uniswap v2 is deployed from 0 to infinity. However, we can assume that Uniswap v2 is simply Uniswap v3 in the limit of infinity range. Specifically, we can convert Uni v3 liquidity into Uni v2 liquidity by considering the relative capital efficiency of Uniswap v3:

In that case, we can convert between the tickTVL of Uniswap v3 with an effective "liquidity at the traded tick" for Uniswap v2 according to:

$$\text{“Effective” tick liquidity} = (\text{Uniswap v2 reserve}) \cdot \left(\frac{1}{1 - \frac{1}{\sqrt{1.003}}} \right)$$

So, for an effective feeRate of 0.3%, the "tick liquidity" of a Uniswap v2 pool would be 668 times smaller than that. The effective feeTier thus becomes:

$$\text{feeTier} = \frac{\text{IV(per year)}}{290273} \sqrt{\frac{\text{reserve0}}{\text{amount0}}} \cdot \sqrt{\Delta T(s)}$$

where reserve0 is the total reserve of token0 in the UniswapV2Pair smart contract and amount0 is the trade size.

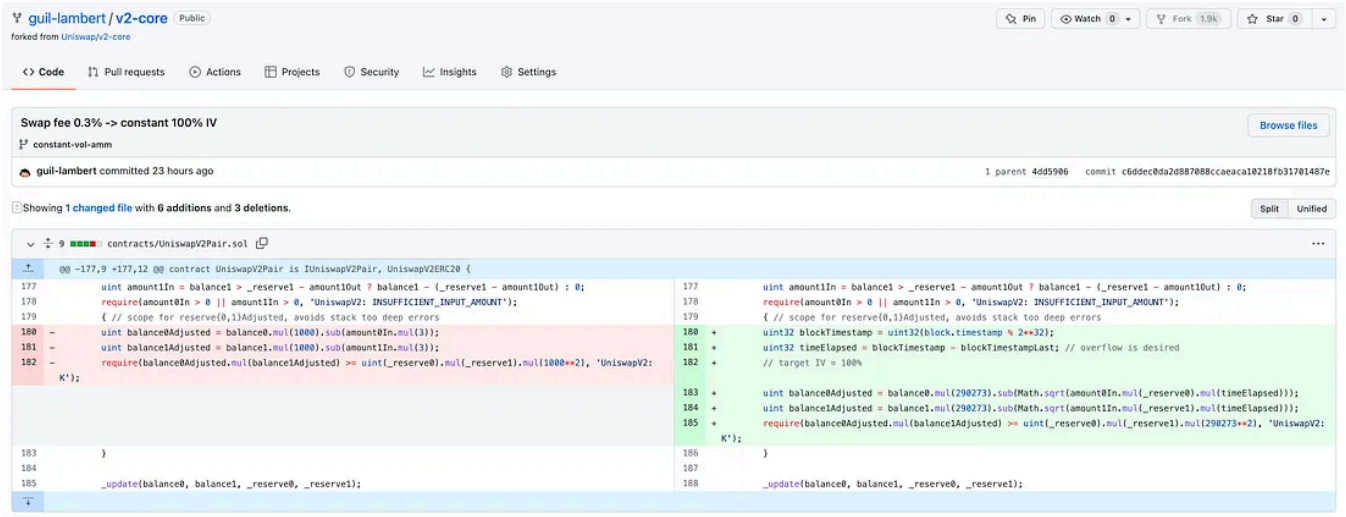
As an example, the current Uni v2 ETH-Dai pool contains 13,886,496 DAI and 3,990 ETH, so a 1 ETH trade would result in a 0.08% fee and a 1000 ETH trade would result in a 0.0026% fee (only 0.026 ETH in fees!).

The reason why these fees are so low is because the effective liquidity at the "traded tick" is only ~6 ETH, which would in turn encourage traders to trade frequently enough so that LPs receive a reward commensurate with a 100% IV asset even though the "per-trade" fee is minuscule.

Forking the Uniswap v2 contract

Implementing this AMM is relatively easy: one only needs to modify the Uniswap v2 smart contracts so that the fee taken on each trade depends on the reserve amount, the traded amount, and the time since the last transaction (blockTimestampLast).

In fact, one does not even need to be a solidity dev to do that, here's the proof (I'm surprised it actually compiles!):



https://github.com/guil-lambert/v2-core/tree/constant-vol-amm

Interested in this novel AMM? We'd love to hear from you! Please reach out to @guil_lambert, @MaxResnick1, or @cryptojesperk on Twitter

Defi + Uniswap + Ethereum + Finance + Volatility +

Follow

Written by Guillaume Lambert

1.7K followers · 22 following

Asst professor in Applied Physics at Cornell. LambertLab.io PI and proud father. Interests: Biophysics, Math, Crypto, and Options (often all at the same time).